D1-82-0349

Also available from author

BOEING SCIENTIFIC RESEARCH LABORATORIES

Uniform Random Number Generators

M. Donald MacLaren

George Marsaglia

Mathematics Research

April 1964

D1-82-0349

UNIFORM RANDOM NUMBER GENERATORS

by

M. Donald MacLaren

and

George Marsaglia

Mathematical Note No. 349

Mathematics Research Laboratory

BOEING SCIENTIFIC RESEARCH LABORATORIES

April 1964

Summary

This paper discusses the testing of methods for generating uniform numbers in a computer--the commonly used multiplicative and mixed congruential generators as well as two new methods. Tests proposed here are more stringent than those usually applied, because the usual tests for randomness have passed several of the commonly used procedures which subsequently gave poor results in actual Monte Carlo calculations. The principal difficulty seems to be that certain simple functions of n-tuples of uniform random numbers do not have the distribution that probability theory predicts.

Two alternative generating methods are described, one of them using a table of uniform numbers, the other one combining two congruential generators. Both of these methods passed the tests, whereas the conventional multiplicative and mixed congruential methods did not.

Introduction

This paper reports the results of extensive testing of random number generators, including two which we propose as improvements on present methods. We carried out this program of testing because random numbers generated by mixed congruential methods gave bad results in a number of Monte Carlo calculations, notably those involving order statistics. The tests suggest that mixed generators are unsatisfactory and that generators of the straight multiplicative type are better but still suspect. For this reason, we propose two generators which are variations on familiar ideas. One combines two congruential generators and the other uses a table of random numbers. Both tested out very well.

By a random number generator, we mean any procedure for producing within a computer a sequence of numbers $U_1, U_2, \ldots$ which is supposed to represent a sequence of independent uniform random variables. Although the U_i are supposed to be random, they are usually generated by a strictly deterministic procedure. Most often the U_i are regarded as integers, and U_{i+1} is computed by

$$U_{i+1} = aU_i + c \mod M, \tag{1}$$

where M is generally taken as 2^n for an n-bit binary machine and 10^n for an n-digit decimal machine. The random number generator is often called "multiplicative" if $c = 0$ and "mixed" if $c \neq 0$.

Generators of this type have been used and favorably reported by many people; an extensive list of references may be found in the survey article [3]. Nevertheless, we have had trouble with mixed generators. This led

us to study the testing of random numbers. We decided that the tests commonly applied to random number generators are not of much value. The tests we discuss in this paper are more stringent than the common tests, but not at all unreasonable.

In attempting to improve on the congruential generators, we tried combining two of them. This gave a generator which seems to be better than either of the two congruential generators, but it has the disadvantage of being slower. We also investigated the possibility of using a table of random numbers. This turns out to be practical on a large computer with buffered input; in fact, for such a machine we recommend this method.

1. A random number generator using a table of random numbers

An obvious alternative to the use of a numerical procedure for generating uniform random numbers is to use some sort of table of random numbers. The big advantage of this is that one should be able to get a table which has none of the undesirable characteristics of the numerically generated sequences. The two main problems with this approach are the time needed to read the table into the computer and the possibility of exhausting the table. In this section we describe a random number generator which uses a table but overcomes these two problems.

To overcome the time problem we use a computer with buffered input. We use two blocks of core storage to hold random numbers, refilling one block from tape while using numbers from the other block. Thus, to get

be considered a set of 485 independent random bits. The whole sequence of random bits was grouped to form the table put on tape.

The number 146 and 485 were chosen to give an efficient conversion from decimal random digits to random bits. It happens that $10^{146} > 2^{485} > .9991 \times 10^{146}$. Therefore, the conversion is 99.9 per cent efficient and obtains almost the maximum of $\log_2 10$ bits per digit.

In the test results summarized in Section 4, the first run used the numbers in the table. The second run used the transformation:

$$U' = (2^{15} + 5)U + 3 \mod 2^{27},$$

and the third run used

$$U' = (2^7 + 1)U + 1 \mod 2^{27}.$$

2. A combination of two congruential generators

The generator described in Section 1 is suitable only for computers with buffered input, and requires tape handling, which may be inconvenient. Therefore, it is still desirable to have some sort of numerical generator. One can, of course, use a congruential generator, but our experience with these has led us to look for generators with better statistical properties.

Suppose the first number U_1 for a congruential generator (1) is picked at random. Then the sequence $U_1, U_2, \ldots$ may be considered a sequence of random variables. Moreover, each U_i will be uniform on $[0,1]$, or rather on the set of numbers in $[0,1]$ which can be represented exactly in the computer. However, the different U_i are not independent, and it turns out that the distribution of an n-tuple $(U_1, \ldots, U_n)$ may

be quite far from the correct distribution. In order to improve the distribution of n-tuples, we propose using two different generators of the type (1) and having one shuffle the sequence produced by the other.

For a test program, we used the generators

$$U_{k+1} = (2^{17} + 3)U_k \mod 2^{35}, \tag{3}$$

and

$$V_{k+1} = (2^7 + 1)V_k + 1 \mod 2^{35}. \tag{4}$$

We let $U_0 = 1$ and $V_0 = 0$. A table of 128 locations in core was filled with the numbers $U_1,\ldots,U_{128}$. Then to generate X_k, the k^{th} random number to be used, we used the first 7 bits of V_k as an index to get X_k from the table. The location of X_k in the table was refilled with the next number from the generator (3).

The time to generate a random number by this method is about twice the time required with a congruential generator. We consider that it is worth suffering this time penalty in order to get a sequence of numbers with better statistical properties. From the test results given in Section 4, we conclude that this generator does indeed have better properties than the congruential generators; refer again to the table and remarks in that section.

3. Testing random numbers

Various procedures have been proposed for testing sequences of random numbers, and extensive references may be found in the survey article [3]. However, most of the proposed tests seem to have little

relevance to the actual uses of random number sequences. Our experience has been that a sequence may pass these tests and yet give a distribution of some simple function $f(U_1,\ldots,U_n)$ of several uniform random variables which is far from the correct distribution. The results reported in Section 4 should convince the reader of this.

The two most commonly used tests, the chi-square test on the distribution of the U_i, and the serial test, are quite useful for weeding out some of the unsatisfactory generators from the outset; they are, in a sense, necessary, but certainly not sufficient. Unfortunately, as usually applied, they are of very limited precision. For example, in [1], Hull, Dobell, and Allard, following the usual practice, divided the unit interval into only ten equal subintervals. In effect, this only tests the first decimal digit of the uniform numbers, and one digit accuracy is low, even for Monte Carlo calculations.

Another limitation of these tests is that they say nothing about the behavior of n-tuples $(U_1,\ldots,U_n)$ for $n > 2$. This is a serious limitation, for it may happen that pairs (U_1,U_2) behave well while triples (U_1,U_2,U_3) behave very badly. The tests on the multiplicative generators illustrate this.

Our philosophy of testing is to test n-tuples $(U_1,\ldots,U_n)$ for as many values of n as possible and to make the tests for each n more stringent than is common. Even so, we still consider that the best practice is to test the sequence to be used for a particular calculation by trying it on a similar problem for which the answer is known.

The tests we actually made were χ^2 tests on the distribution of the random numbers, pairs of random numbers, triples of random numbers, and various simple functions of several random numbers. The idea of testing simple functions of several random numbers was to have some sort of test of the behavior of n-tuples $(U_1,\ldots,U_n)$ for $n > 3$ and to have tests of pairs and triples which would be in some ways more stringent than the basic tests.

Some of the tests were based on the distribution of the maximum and minimum of n uniform variables. These were chosen as the easiest of the order statistics of n uniform variables; our previous dissatisfaction with order statistics generated by some of the congruential generators led us to include these in our routine tests of any generators. We also routinely test the distribution of the sum of 2, 3, and 4 uniform variables, as these distributions are useful in certain procedures for generating arbitrary random variables.

All the tests had the same general form. A sequence of n variables $X_1,X_2,\ldots$ was computed from the sequence of uniform numbers. If the uniform numbers were actually independent uniform random variables, the X_i would be independent identically distributed random variables, perhaps multi-dimensional. The range of the X_i was divided into m cells of equal probability p and the number of occurences k_i in each cell counted. The χ^2-statistic

$$\chi^2 = \sum_{i=1}^{m} \frac{(k_i-np)^2}{np}$$

was computed. We then computed the corresponding percentile, i.e. the probability that for a random variable with the hypothetical distribution the value of χ^2 would exceed the observed value. The percentiles were

rounded to two figures except from some very small values. These percentiles are tabulated in Section 4. Where ϵ appears, it means the percentile was less than .02 per cent.

All the tests were made simultaneously using the same sequence $U_1, U_2, \ldots$ Thus the results of the various tests are not independent. Moreover, as a programming convenience, we took the uniform numbers in sets of ten. From a set of ten we obtained five pairs for the pairs test, three triples for the triples test, and one each of the various one-dimensional variables. Thus from a set of ten uniform numbers, only one pair is used for the test of the maximum of two uniforms, one triple for the maximum of three, etc. The exact details of this are given below.

Three separate runs were made for each generator, and the different runs were independent. In fact, each run started at the point in the sequence $U_1, U_2, \ldots$ where the last left off.

We conclude this section by summarizing the details of each test.

Uniformity. This was included for the sake of completeness. The unit interval was divided into 100 equal cells. For each run 10,000 uniform numbers were used, the exact sequence being $U_1, U_{11}, U_{21}, \ldots$ The occurences in each cell were counted, and the χ^2 statistic computed.

Pairs. Successive pairs of uniform numbers were taken as the coordinates of a point in the unit square. The unit square was divided into 400 equal cells. For each run a total of 50,000 pairs was generated.

Triples. Successive triples of uniform numbers were taken as the coordinates of a point in the unit cube. However, every tenth uniform number was skipped. The cube was divided into 1000 equal cells. For each run a total of 30,000 triples was generated.

Maximum of n. This test was made for $n = 2, 3, 4, 5, 10$. Let F denote the distribution function of the maximum of n independent uniform random variables. Then if $V_1, V_2, \ldots, V_n$ are independent uniform random variables, $F(\max(V_1, V_2, \ldots, V_n))$ is uniform on $[0,1]$. We calculated $F(\max(V_1, V_2, \ldots, V_n))$ for n-tuples from the sequence $U_1, U_2, \ldots$ The unit interval was divided into 100 equal sub-intervals. For each run, we used 10,000 n-tuples. The sequence of n-tuples actually used was $(U_1, \ldots, U_n)$, $(U_{11}, \ldots, U_{10+n}), \ldots$

Minimum of n. This test was the same as that for the maximum of n except that we took the minimum of n-tuples $(V_1, \ldots, V_n)$ and used the corresponding distribution function.

Sum of n. Again this test was similar to that for the maximum of n. We tested to see if $G(V_1 + \cdots + V_n)$ was uniform, where G is the distribution function of the sum of n uniform random variables.

4. Test results and General Remarks

Besides the generators described in Sections 1 and 2, we tested five congruential generators. For mixed generators, we chose

$$U_{k+1} = 101U_k + 1 \mod 10^{10}, \quad U_0 = 0, \tag{5}$$

and

$$U_{k+1} = (2^7 + 1)U_k + 1 \mod 2^{35}, \quad U_0 = 0. \tag{6}$$

These have been favorably reported on in [1] and [2]. To represent multiplicative generators, we chose

(7) $$U_{k+1} = (100003)U_k \qquad \text{mod } 10^{10}, \qquad U_0 = 1,$$

and

(8) $$U_{k+1} = (2^{17} + 3)U_k \qquad \text{mod } 2^{35}, \qquad U_0 = 1.$$

These multipliers insure the maximum possible period of 5×10^8 and 2^{33} respectively. They follow the suggestion in [4] that the multiplier be close to $\sqrt{M}$, where M is the modulus. Finally, we tested Lehmer's original method for the Eniac.

(9) $$U_{k+1} = 23U_k \qquad \text{mod } 10^8 + 1, \qquad V_0 = 47594118.$$

This was the first congruential generator proposed, and it is still widely used.

The table reports the results of 33 tests made on each of the seven generators. These do not include all the tests actually performed, but the results tabulated do give a fair picture of the behavior of the various generators. The quantity given in the table is the percentile for the appropriate chi-square distribution. Thus, the 33 values for each generator should behave somewhat like a set of 33 numbers chosen uniformly from the unit interval. There may be occasional small or large values, but any preponderance of such values must be suspect.

On the basis of these tests, which may be viewed as preliminary, but certainly indicative, the two mixed congruential methods and Lehmer's

original method are not satisfactory; the two multiplicative methods are suspect, while the methods based on a stored table or on mixing of two congruential generators appear to be satisfactory. It is true that the results of the pairs test for the stored-table generator might be viewed with suspicion. Note, however, that for neither the stored-table generator nor the combination generator do any epsilons, denoting a percentile less than .02 per cent, appear in the table. Moreover if the table of random numbers used in the stored table generator should turn out to be unsatisfactory, one can always replace it with a better table. We expect that an excellent table of random numbers can be prepared by using a sequence of congruential generators to repeatedly shuffle either the set of numbers produced by a congruential generator or one of the standard tables of random digits.

In conclusion, we would like to point out that with the high speeds of modern computers it is no longer important to look for the fastest possible random number generators. All generators are reasonably fast, even the combination of two congruential generators takes only 56 microseconds per random number in the 7094 computer. Therefore, we should look for the generator which produces the most suitable sequences of random numbers. We expect that these will turn out to be generators using a table of random numbers.

	Generator using a table of random numbers			Combination of two congruential generators			$U_{k+1} = (2^{17}+3)U_k$ Mod 2^{35}			$U_{k+1} = (10^5+3)U_k$ Mod 10^{10}		
Test	Run 1	Run 2	Run 3	Run 1	Run 2	Run 3	Run 1	Run 2	Run 3	Run 1	Run 2	Run 3
Uniformity	59	49	39	1.9	7.7	33	78	17	60	68	89	33
Pairs	2.8	11	14	85	85	60	73	44	61	92	26	62
Triples	5.3	89	95	8.1	49	46	ϵ	ϵ	ϵ	.38	ϵ	ϵ
Sum of 2	92	51	22	27	29	84	94	68	85	84	27	4.0
Sum of 4	15	58	15	11	84	77	9.9	20	40	55	34	69
Maximum of 2	78	66	36	21	48	29	92	76	57	7.6	41	1.8
Maximum of 5	68	97	67	98	15	33	41	21	30	94	7.3	65
Minimum of 3	69	97	85	100	22	66	76	39	54	1.4	87	54
Minimum of 10	83	87	22	18	22	31	40	26	27	2.3	73	13

The tabled value is the percentile for the appropriate chi-square distribution. Where ϵ appears, it means the percentile was less than .02 per cent.

	$U_{k+1} = (2^7+1)U_k + 1$ Mod 2^{35}			$U_{k+1} = 101U_k + 1$ Mod 10^{10}			Lehmer's method $U_{k+1} = 23U_k$ Mod $10^8 + 1$		
Test	Run 1	Run 2	Run 3	Run 1	Run 2	Run 3	Run 1	Run 2	Run 3
Uniformity	99	91	35	11	44	9.3	75	63	60
Pairs	ε	ε	ε	ε	ε	ε	ε	ε	ε
Triples	70	34	13	2.3	4.1	16	ε	ε	ε
Sum of 2	7.6	20	75	92	58	26	72	15	8.7
Sum of 4	41	26	35	50	55	50	4.1	12.6	.11
Maximum of 2	5.9	17	9.6	ε	ε	2.9	ε	ε	ε
Maximum of 5	.39	ε	ε	ε	ε	.95	ε	ε	ε
Minimum of 3	3.0	2.7	ε	ε	ε	ε	ε	ε	ε
Minimum of 10	ε	.10	ε	ε	ε	ε	ε	ε	ε

The tabled value is the percentile for the appropriate chi-square distribution. Where ε appears, it means the percentile was less than .02 per cent.

References

1. J. L. Allard, A. R. Dobell, and T. E. Hull, "Mixed Congruential Random Number Generators for Decimal Machines," J. Asso. Comp. Mach. 10 (1963) 131-141.

2. T. E. Hull and A. R. Dobell, "Mixed Congruential Random Number Generators for Binary Machines," J. Asso. Comp. Mach. 11 (1964) 31-40.

3. T. E. Hull and A. R. Dobell, "Random Number Generators," SIAM Review 4 (1962) 230-254.

4. International Business Machines Corporation, Random Number Generation and Testing, Reference Manual C20-8011 (New York, 1959).